

# 基于 Memcached 的日历搜索引擎系统优化设计与实现

薛献鹏<sup>1</sup>, 彭明田<sup>1,2</sup>, 贺怀清<sup>1</sup>

(1. 中国民航大学 计算机科学与技术学院, 天津 300300; 2. 中国民航信息网络股份有限公司 研发中心, 北京 100176)

(xxphddz@163.com)

**摘要:** 研究基于航信的日历搜索引擎(CS)系统, 针对日历搜索引擎系统计算量大、响应速度慢、重复计算的问题, 提出了一种利用 Memcached 对计算单元结果进行缓存的方法, 基于此方法对系统架构进行重新设计, 并对日历搜索引擎系统进行性能优化。实验结果显示该优化方案减少了系统响应时间, 使系统性能得到了大幅度的提升, 为民航运价领域中日历搜索引擎系统的优化提供了方法和理论支持。

**关键词:** 日历搜索引擎; Memcached; 最低运价搜索; 民航

**中图分类号:** TP319 **文献标志码:** A

## Optimal design and implementation of calendar shopping system based on Memcached

XUE Xian-peng<sup>1</sup>, PENG Ming-tian<sup>1,2</sup>, HE Huai-qing<sup>1</sup>

(1. School of Computer Science and Technology, Civil Aviation University of China, Tianjin 300300, China;

2. Research and Development Center, TravelSky Technology Limited, Beijing 100176, China)

**Abstract:** Concerning the problem of large computation, slow response and repeated computation in travel sky-based Calendar Shopping (CS) system, an efficient method was proposed to cache calculation results of unit in this paper. The system architecture was redesigned and the performance of calendar shopping system was optimized. The experimental results show that the presented method can reduce the system response time and improve the performance of the system significantly, also it provides the method and theoretical support for calendar shopping system in civil aviation field.

**Key words:** Calendar Shopping (CS); Memcached; Lowest Fare Search (LFS); civil aviation

## 0 引言

日历搜索引擎(Calendar Shopping, CS)产品是中国民航信息网络股份有限公司推出的用于支持航空公司直接销售渠道的结合可用座位的专用系统。引擎一次查询即可查询到指定时间范围内每天的最低可用运价, 从而避免了用户在进行机票选择时多次查询、反复比较。这一方面节省了用户的时间, 增加了顾客的购票体验; 另一方面减轻了多次查询给服务器带来的压力。在国外也提供了类似的最低运价搜索(Lower Fare Search, LFS)服务, 在美国, 以 Carl G. Demarcken 为首的开发团队主要基于并行技术和缓存机制对其进行性能优化和系统改造<sup>[1-4]</sup>。本课题在中航信已有日历搜索引擎产品的基础上, 基于 Memcached 技术对其进行优化设计和实现。

本文首先对 CS 架构和 Memcached 的工作原理进行了简单介绍; 然后基于 Memcached 对 CS 系统进行优化设计和实现; 最后给出了实验结果及分析。实验结果表明, 该优化方案提高了民航运价领域中 CS 的响应速度并降低了系统资源消耗, 为分布式计算领域的数据共享提供了一种借鉴方法。

## 1 相关工作

### 1.1 CS 系统

CS 系统整体架构如图 1 所示, CS 是基于 BEA Tuxedo 架

构的 Tuxedo Service。它需要调用 OPEN AV 系统获取 AV 信息, 通过分布式 Oracle 数据库获得必要的运价和规则信息, 航空公司直销网站通过航信互联网订座引擎(Internet Booking Engine, IBE)接口接入 CS 系统。网站根据客户要求构造请求, 并通过 IBE 平台将其转发给 CS 系统。IBE 获取 XML 形式的响应结果并转发回调用网站, 由该网站对响应结果进行解析, 为客户进行个性化的展示。CS 处理流程如图 2 所示。

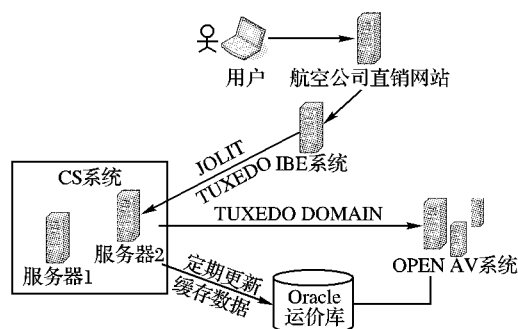


图1 CS系统整体架构

### 1.2 Memcached 工作原理

Memcached 是一个高性能的分布式内存对象缓存系统, 允许不同主机上的多个用户同时访问。很多大型 Web 应用程序包括 Facebook、Youtube、Yahoo、Wikipedia 等都在使用 Memcached 来支持每天数亿级的页面访问<sup>[5]</sup>。

收稿日期: 2010-07-23。

**作者简介:** 薛献鹏(1984-), 男, 河北邯郸人, 硕士研究生, 主要研究方向: 民航信息系统架构、民航信息系统设计; 彭明田(1967-), 男, 安徽宿州人, 高级工程师, 硕士, 主要研究方向: 民航信息系统架构、民航信息系统设计; 贺怀清(1969-), 女, 吉林白山人, 教授, 博士, CCF 会员, 主要研究方向: 图形图像与可视化、民航相关信息可视分析、民航简化商务相关标准研究、语音识别。

Memcached 中保存的数据都存储在 Memcached 内置的内存空间中,当内存达到指定值之后,就基于最近最少使用(Least Recently Used, LRU)算法自动删除不使用的内存。它使用预申请和分组的方式管理内存空间,允许多个服务器通过网络形成一个大的 Hash 表。Memcached 以守护程序方式运行于一个或多个服务器中,随时接受客户端连接操作。每个被存取的对象都有一个唯一的标识符 key,存取操作均通过这个 key 进行。

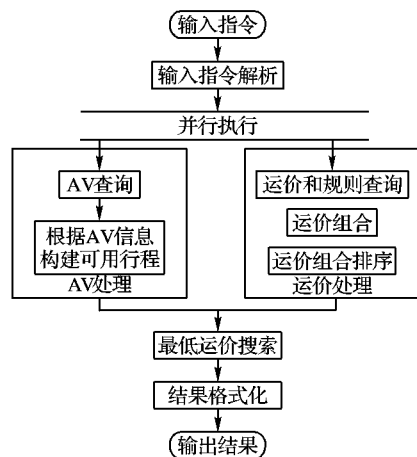


图2 CS处理流程

## 2 CS系统优化设计与实现

### 2.1 优化思路

多次实验表明,相关数据的获取与解析、最低运价搜索占整个处理过程的99%以上。最低运价搜索就是找到每个运价单元的最低运价。在实验环境中,对于计算量最大的请求,最低运价搜索部分所占时间比重可以高达58%。由于请求对象经常重复,为了支持日益增长的访问量,可以通过缓存计算结果的方式来减少重复计算,从而提高整个系统的并发量和吞吐量。

### 2.2 优化方案

Cache是用来解决最耗性能部分的数据访问的,也就是访问频率高且响应时间长的计算请求。经综合考虑,对于热点数据,即热点航线和热点时间段内的计算请求,进行主动装载和预先缓存并定时更新,以解决命中率低的问题。对于其他请求进行惰性装载(仅当请求到来时),对计算结果进行短时间缓存。至于缓存基本单元的选择,基于复用性考虑,选定缓存基本单元=单程每天的最低价或一个来回程组合的最低价,所谓的来回程就是从出发地点至目的地点并按原航程返回原出发地点的航程。由于现阶段只能支持 $\pm 3$ 天的情况,一个单程 shopping 结果可以存成7个Cache键值对,一个来回程 shopping 结果可以存成49个Cache键值对。

### 2.3 设计和实现

Cache系统采用Client/Server模式。Server端启动若干个Memcached进程。Client按照功能可划分为主动加载、惰性加载和Memcached缓存客户端部分。

#### 2.3.1 主动加载管理模块

该部分包含一个Memcached\_load进程,Memcached\_load系统框架如图3所示。当更新时机到来时,根据预先设定的热点数据构造请求对象并调用CS服务,根据响应结果构造

缓存基本单元键值对并更新缓存信息,该进程主要包含以下几个模块。

1) 解析模块。本模块负责读取并解析配置文件,进行模块初始化,设定更新策略,确定热点数据,进行首轮热点数据主动加载等。

2) 请求构造模块。根据热点数据对象构造请求对象,格式化符合CS接口协议的XML字符串。

3) 请求/响应模块。调用CS服务(配置请求使CS走普通计算路线),并获取响应结果。

4) 数据更新模块。解析响应结果并将运价单元结果序列化成字符串,组成Cache键值对,并更新到Memcached。

5) 中心管理模块。调用各个模块协同工作。

当进程启动时,进程首先读取并解析配置文件,进行首轮热点数据装载,然后系统进入更新循环之中。主动加载循环流程如图4所示。

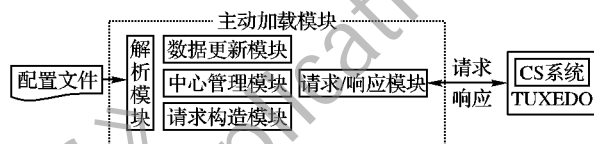


图3 Memcached\_load系统框架

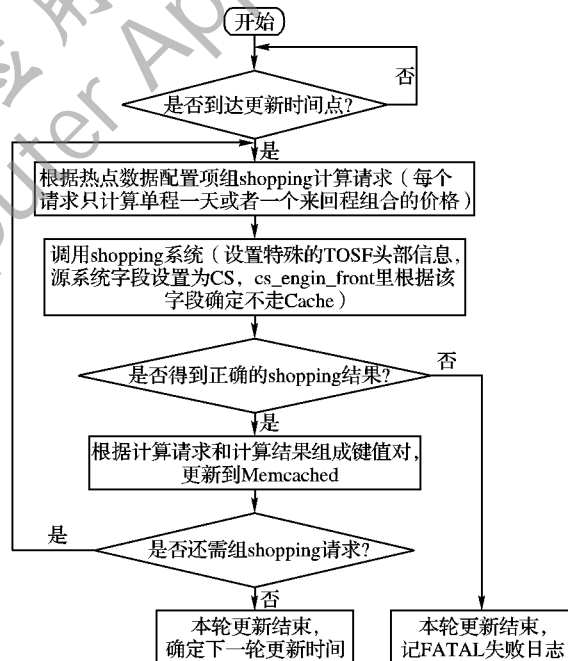


图4 主动加载更新流程

#### 2.3.2 惰性加载管理模块

该部分集成在CS系统框架中,以LazeLoadMng类封装其功能。当非热点请求缓存基本单元键值对更新到Memcached之后,该模块负责惰性管理信息的更新,并在删除时机到来时启动删除失效记录的操作。

该模块维护着两个全局map:mapOWInfos和mapRTInfos,其分别用于单程和来回程的惰性信息维护。对于mapOWInfos的每一个键值对,键记为Key,值记为Value。Key由航空公司、起始地、目的地字符串组合而成;Value包含一个记录数组,记录着出发日期范围和该记录更新时间,类型为OWInfo。Key和记录项组合起来标识着一个请求对象。当两个记录属于同一个Value,并且出发日期相同时,认为这两个记录标识着同样的请求。mapRTInfos维护着来回程的惰性信

息,具有同 mapOWInfos 相似的结构和性质,如图 5 所示。

对于一个非热点数据的单程请求,当普通计算流程完成并更新 Memcached 之后,首先需要对单程惰性信息进行更新操作,该操作通过 LazeLoadMng 的 add 方法提供。

LazeLoadMng 类 add 的单程部分算法。

输入 mapOWInfos, 当前时间 currentTime, 请求对象 Req。

输出 更新后的 mapOWInfos。

1) 字符串 key = 请求的航空公司 + 起始地 + 目的地;

2) 找出 mapOWInfos 中 key 所对应的键值 value。

if (value 存在)

{ 获得 value 的记录数组 vector < OWInfo > vecOWInfo;

整数 i = 0;

for(i = 0; i < vecOWInfo.size(); i++)

{ if (vecOWInfo[i].出发时间 == 请求的出发时间)

{ vecOWInfo[i].setCreateTime( currentTime );

/\* 将其有效时间更新为当前时间 \*/

break;

}

}

if (i = 0; i < vecOWInfo.size()) //没有找到相同的记录

{ 根据 Req 构造新的 OWInfo 对象 newOWInfo;

vecOWInfo.push\_back( newOWInfo );

}

}

else

{ 构造新的 Value 对象 newValue;

根据 Req 出发时间构造新的 OWInfo 对象 newOWInfo;

newOWInfo 添加到 newValue 的记录数组中;

键值对( key, newValue) 添加到 mapOWInfos 中;

}

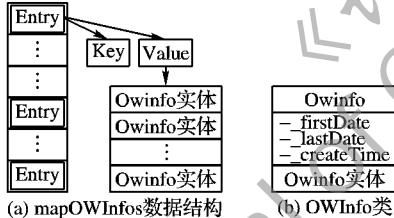


图 5 mapOWInfos 数据结构和 OWInfo 类

为了防止 mapOWInfos 中充斥着失效缓存信息的记录信息,需要按照一定机制删除失效记录,若到达删除时机则开始新一轮的删除工作。该操作通过 LazeLoadMng 的 update 方法提供。

LazeLoadMng 类 update 算法的单程部分如下:

输入 mapOWInfos;

输出 mapOWInfos。

map < string, Value > :: iterator iter;

for( iter = mapOWInfos.begin(); iter = mapOWInfos.end(); iter++)

{

获得 iter->second 指向的 Value 对象 value;

获得 value 的记录数组 vector < OWInfo > vecOWInfo;

for( int i = 0; i < OWInfos.size(); )

{

if ( OWInfos[ i ] 无效)

{

OWInfos.remove( i );

continue;

}

i++;

}

}  
if( OWInfos.empty() )

{

删除该 iter 所指向对象, 并从 mapOWInfos 中将其删除

}

}

来回程的记录更新和删除算法与单程类似,在此不予赘述。惰性装载方式作为主动装载方法的有效补充,不仅保持了主动装载命中率高的优点,针对热点数据覆盖面不广、热点数据之外频繁请求不予缓存支持的弱点,特意为热点数据外突发性请求提供了缓存服务。

### 2.3.3 Memcached 用户客户端

该部分功能以类 MemcacheHelper 包装。负责 CS 系统与 Memcached 服务器交互及相关处理。该类实现了三种逻辑功能:1)判断 cache 中是否存在缓存结果;2)缓存基本单元/反序列化;3)缓存基本单元的获取和添加。

在一次 CS 请求处理中,首先判断 Cache 中是否存在缓存结果,若缓存结果全部存在,则组结果并退出处理流程;否则,将走普通计算流程并将未命中的缓存基本单元缓存到 Memcached 中。在惰性管理信息更新之后,若删除时机已到,则开始新一轮的删除。Memcached 中缓存的基本单元是文本串形式,这就需要在缓存基本单元之前把基本单元序列化成年字符串,获取基本单元之后把字符串反序列化为基本单元对象。

### 2.4 系统配置考虑因子

预先确定的热点航线数目越多,命中率越高,同时主动更新耗时也会更久。当数据量超过计算机物理内存时,就需要在硬盘和内存中进行频繁的数据交换。航线个数一定时,良好选择可以最大限度增加系统命中率,主要通过经验和统计两种方式选择热点数据。更新时间间隔必须大于一次主动加载所耗时间,但时间间隔太久又会使计算结果“脏”的几率增大。

## 3 实验结果与分析

实验环境:4 个逻辑 CPU, CPU 为 Intel Xeon E7450

2.40 GHz;内存为 4 GB;操作系统为 Linux x(86\_64)。

通过配置文件设定国航的北京到上海的最近两个月的数据为热点数据,当前日期为 2010-06-05。分别构造三种请求类型:1)热点数据请求。航空公司国航,航线北京到上海,出发日期范围为 2010-06-07 至 2010-06-13,返程日期为 2010-07-07 至 2010-07-13;2)非热点数据请求。航空公司国航,航线北京到成都,出发日期范围为 2010-06-07 至 2010-06-13,返程日期为 2010-07-07 至 2010-07-13;3)部分命中请求。航空公司国航,航线北京到成都,出发日期范围为 2010-06-10 至 2010-06-17,返程日期为 2010-07-08 至 2010-07-14。配置一个 Memcached 服务器。

首先验证主动装载的优化效果,分别向优化前和基于 Memcached 优化的系统发出热点数据请求。同步方式调用效果如图 6(a)所示,异步方式如图 6(b)所示。从中可以看出,对于热点数据的请求,无论是以同步还是异步方式发出,基于 Memcached 优化的 CS 系统响应时间都大大缩短。图 6(c)中,对于同步调用响应时间比普通可达 13 倍以上,最大可以

达到 13.80。

系统通过惰性装载的方式对非热点的数据请求进行性能优化。当非热点数据请求首次来临时,缓存基本单元全部未命中,系统进行普通计算流程并将缓存基本单元添加到 Memcached 中。为了模拟首次请求来临的情况,修改处理过程使获取缓存基本单元失败,这样就会造成缓存全部未命中的情况。100 次实验效果如图 7(a) 所示,Memcached 计算流程较普通计算流程时间略有增加,这是因为 Memcached 计算流程除普通计算流程之外,还需要将缓存数据增加到 Memcached 中。当非热点数据请求在短时间内再次来临时,缓存基本单元会全部命中,异步调用效果如图 7(b) 所示,同热点数据具有相似的优化效果。

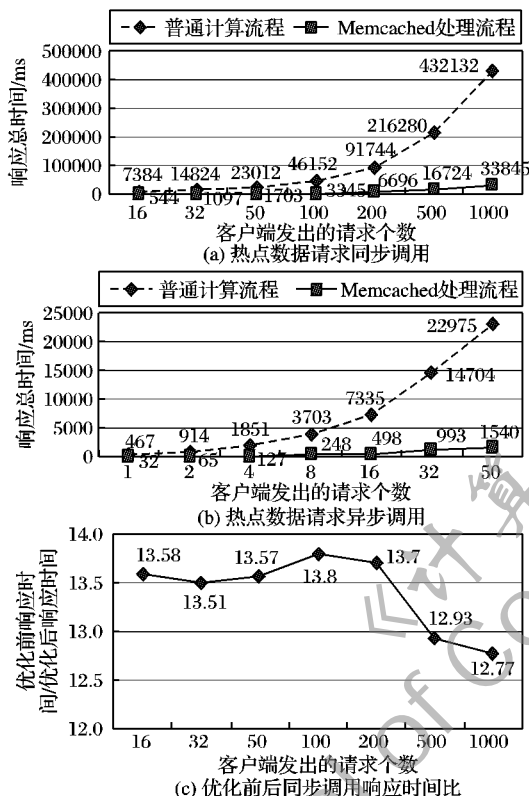


图6 热点数据请求调用效果及响应时间比较

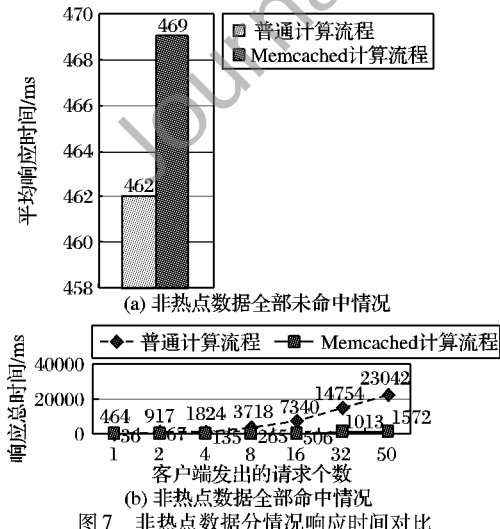


图7 非热点数据分情况响应时间对比

图7对缓存基本单元全部命中和没有命中两种情况分别进行了对比实验,下面对缓存基本单元部分命中的情况进行实验对比。对于优化之前的系统,假设:

$a = \text{计算部分时间} / \text{处理时间}$

则当实验次数逐渐增多时:

$$1 - a < \frac{\text{优化之后的平均处理时间}}{\text{优化之前的平均处理时间}} \leq 1$$

对于来回程,例如:北京—上海—北京。当进行一次非热点数据请求调用之后,进行部分命中请求实验,该请求共有  $7 \times 7 = 49$  个运价单元,其中缓存基本单元命中个数  $4 \times 6 = 24$ ,100 次实验情况如图 8 所示。优化之后的响应时间相当于优化之前的 81.7%,系统性能也得到了一定程度的优化。

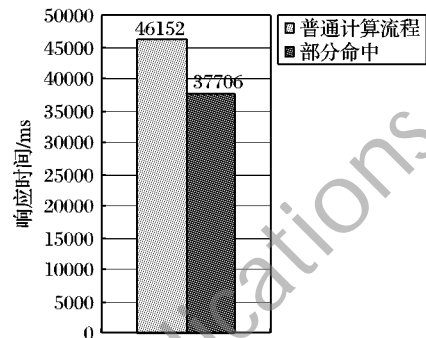


图8 缓存基本单元部分命中情况对比

综上所述,除了对缓存基本单元全部未命中系统性能略有下降外,系统性能都有不同程度的提升。

#### 4 结语

本文基于 Memcached 对 CS 系统进行性能优化。通过缓存基本单元主动装载和惰性装载相结合的优化方式,使 Memcached 的优点得到了淋漓尽致的体现,同时为分布式计算领域的数据共享提供了一种借鉴方法。实验结果表明,Memcached 使系统性能得到了大幅度提高,也会极大地提高顾客的购票体验。当然优化方法尚有不足之处:对于主动装载,热点数据基于经验确定,不能基于统计信息动态调整热点数据;对于惰性装载,只能对缓存基本单元进行短时间的缓存,对于访问频率较高的非热点数据请求不能基于请求情况预先进行长时间的缓存。这是本课题下一步要考虑的,同时结合并行技术对其进行进一步优化。

#### 参考文献:

- [1] DEMARCKEN C G. Dividing a travel query into sub-queries [EB/OL]. (2004-01-01) [2010-04-24]. <http://www.freepatentsonline.com/y2004/0078251.html>.
- [2] DEMARCKEN C G, ROYAN J A. Query widening for query caches for travel planing system [EB/OL]. (2004-01-01) [2010-04-28]. [http://www.wipo.int/pctdb/en/wo.jsp?KEY=04%2F109558.041216%26ELEMENT\\_SET%3DDECL&IA=US2004018683&DISPLAY=CLAIMS](http://www.wipo.int/pctdb/en/wo.jsp?KEY=04%2F109558.041216%26ELEMENT_SET%3DDECL&IA=US2004018683&DISPLAY=CLAIMS).
- [3] DEMARCKEN C G. Support for flexible travel planning [EB/OL]. (2008-01-01) [2010-04-24]. <http://www.freepatentsonline.com/y2008/0167906.html>.
- [4] DEMARCKEN C G, BOYAN J A. Flexible-date travel queries [EB/OL]. (2008-01-01) [2010-04-24]. <http://www.google.com.hk/patents?hl=zh-CN&lr=&vid=USPATA-PP12044028&id=jrtyAAAAEBAJ&oi=fnd&dq=Flexible-date+travel+queries&printsec=abstract#v=onepage&q&f=false>.
- [5] Memcached 原理和使用详解 [EB/OL]. [2010-01-01]. <http://blog.csdn.net/heiyeluren>.